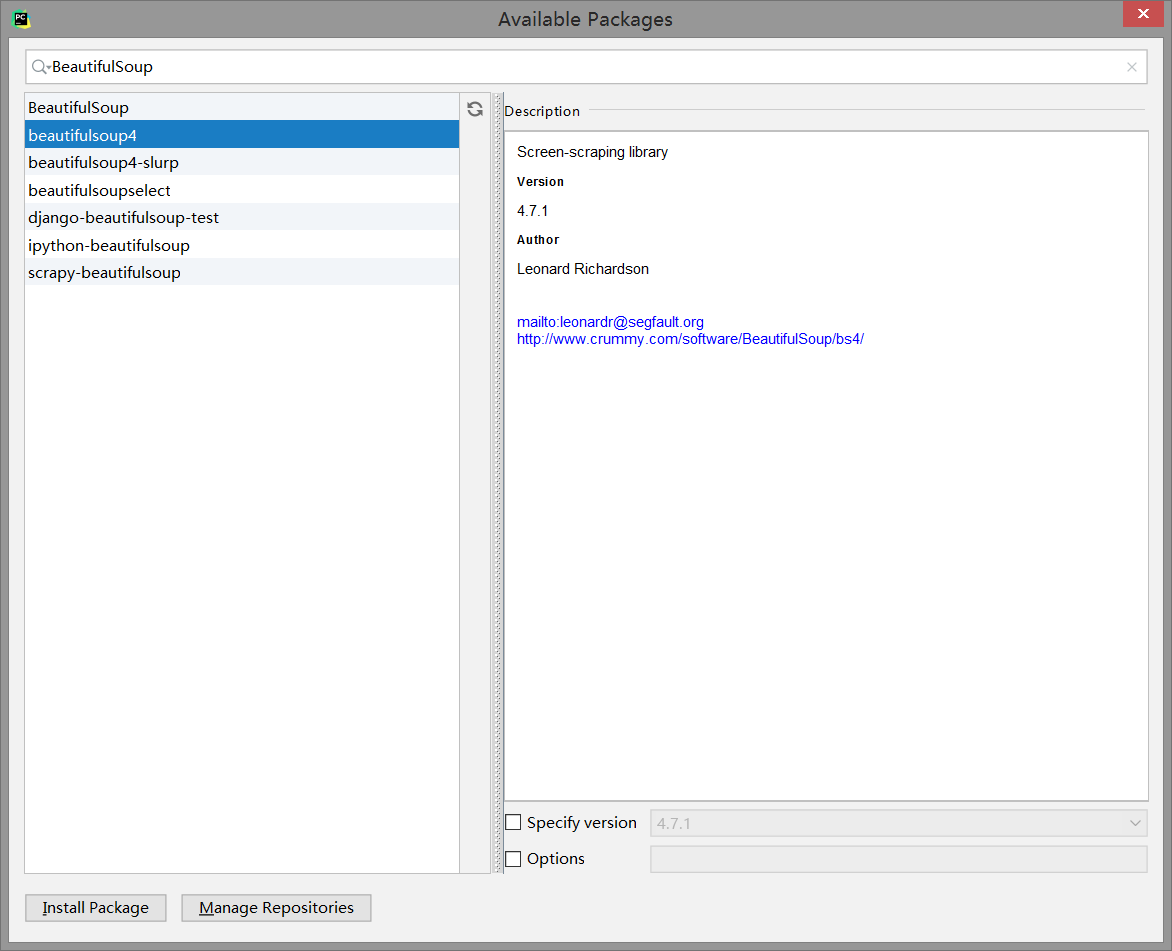
# BeautifulSoup 教程

## 安装(Python3.7)

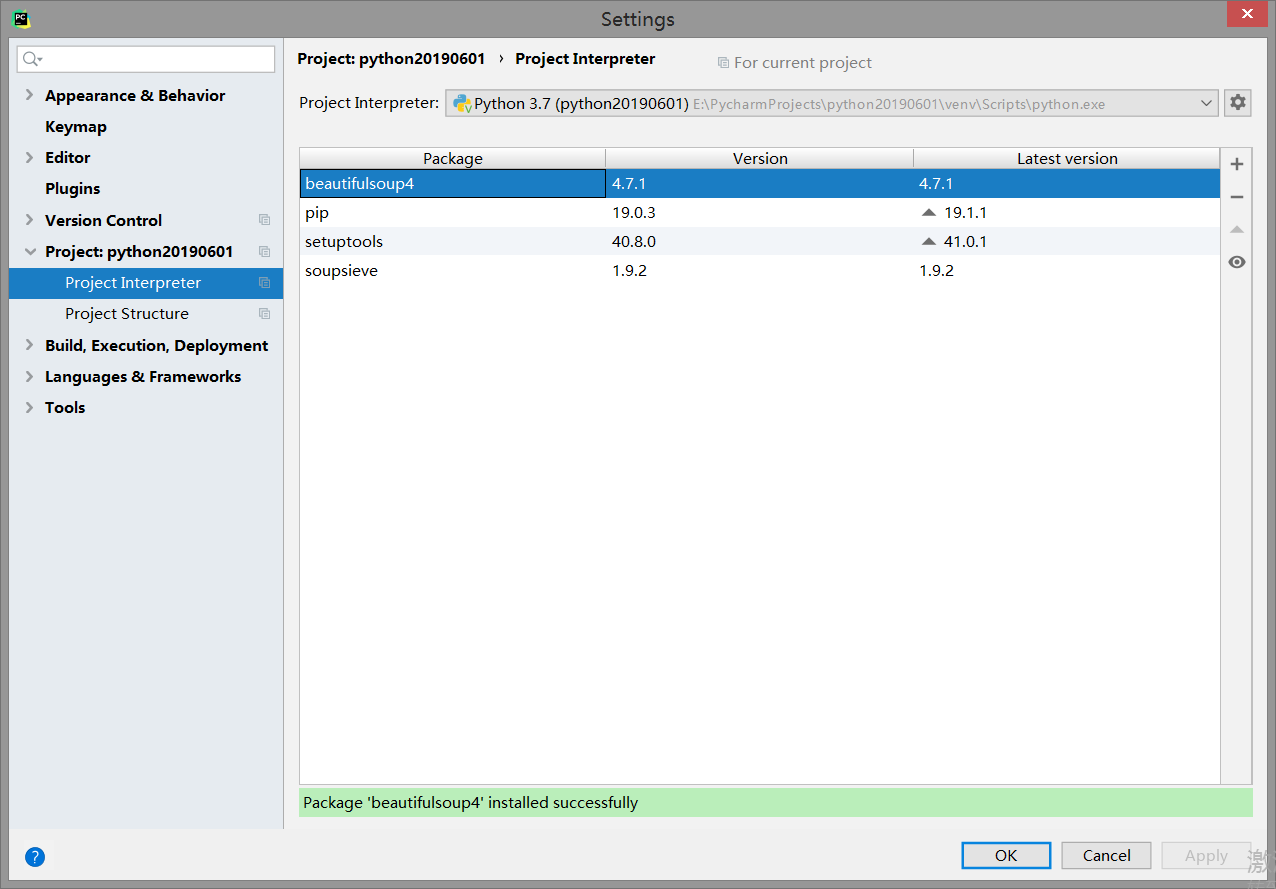
使用pip install beautifulsoup4，或者使用pycharm安装

点击File | Settings | Project: python20190601 | Project Interpreter

点击右上角加号，搜索beautifulsoup4，点击Install Package(注：如果速度慢，提示连接超时，可以换国内镜像，然后重试)



安装成功如图所示



## 安装lxml

lxml：python的HTML/XML的解析器

使用 pip install lxml 命令进行安装，或者使用pycharm工具进行安装

如果安装失败，可以手动选择版本

cmd模式下

python

>>>import pip.\_internal

>>>print(pip.\_internal.pep425tags.get\_supported())

根据内核版本选择合适的版本

https://www.lfd.uci.edu/~gohlke/pythonlibs/#lxml

## 导入第三方库

from bs4 import BeautifulSoup

## BeautifulSoup 对象

创建 beautifulsoup 对象：

soup = BeautifulSoup(html,'html.parser')

还可以用本地 HTML 文件来创建对象：

soup1 = BeautifulSoup(open('index.html'))

打印一下 soup 对象的内容，格式化输出：

print soup.prettify()

输出结果，格式化打印出了它的内容，这个函数经常用到。

Beautiful Soup将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象,所有对象可以归纳为4种:

一、Tag：Tag就是 HTML 中的一个个标签

Tag就是 HTML 中的一个个标签，例如：

<title>The Dormouse's story</title>

用 BeautifulSoup 可以很方便地获取 Tags，这种方式查找的是在所有内容中的第一个符合要求的标签。

soup.title

soup.head

soup.a

soup.p

type(soup.a)

soup.name

soup.a.name

soup.attrs

soup.p.attrs #在这里，我们把 p 标签的所有属性打印输出了出来，得到的类型是一个字典。

soup.p['class'] #单独获取某个属性

soup.p.get('class') ##单独获取某个属性 跟上面一样的

NavigableString：得到了标签的内容用 .string 即可获取标签内部的文字

Soup.p.string##一个tag可以调用.string获取内部文字，类型是

<class 'bs4.element.NavigableString'>

BeautifulSoup：BeautifulSoup 对象表示的是一个文档的全部内容.大部分时候,可以把它当作 Tag 对象，是一个特殊的 Tag，我们可以分别获取它的类型

Comment：Comment 对象是一个特殊类型的 NavigableString 对象，其实输出的内容仍然不包括注释符号。我们使用.string获取注释对象，输出时跟普通字符串没有区别，我们可以通过类型判断一下，然后输出。注释类型如：<class 'bs4.element.Comment'>

## 遍历文档树

tag 的 .content 属性可以将tag的子节点以列表的方式输出

tag 的 .children属性可以将tag的子节点以生成器方式输出，通过for循环遍历获取每个元素

tag 的 . descendants 属性可以将tag的子孙节点以生成器的方式输出

tag的 .string属性可以获取标签内的值，.strings属性可以获取标签内的多个值

tag的 .parent属性可以获取父标签

tag的 .next\_sbling属性可以获取下一个兄弟节点

tag的 . previous\_sibling属性可以获取下一个兄弟节点

tag的 .next\_siblings 和 .previous\_siblings可以获取后边所有，前边所有兄弟节点

## 搜索文档树

find\_all( **name , attrs , recursive , text , \*\*kwargs** )

name参数

* 1. 传入字符串

如soup.find\_all('b')

* 1. 传入正则表达式

soup.find\_all(re.compile("^b")):

* 1. 传入列表

soup.find\_all(["a", "b"])

* 1. 传入True：可以匹配任何值,下面代码查找到所有的tag,但是不会返回字符串节点：

keyword参数

soup.find\_all(id='link2')

soup.find\_all("a", class\_="sister")

recursive 参数：调用tag的 find\_all()方法时,Beautiful Soup会检索当前tag的所有子孙节点,如果只想搜索tag的直接子节点,可以使用参数 recursive=False：

soup.html.find\_all("title", recursive=False)

find( name , attrs , recursive , text , \*\*kwargs )

它与 find\_all() 方法唯一的区别是 find\_all() 方法的返回结果是值包含一个元素的列表,而 find() 方法直接返回结果

find\_parents() find\_parent()

find\_all() 和 find() 只搜索当前节点的所有子节点,孙子节点等. find\_parents() 和 find\_parent() 用来搜索当前节点的父辈节点,搜索方法与普通tag的搜索方法相同,搜索文档搜索文档包含的内容

find\_next\_siblings() find\_next\_sibling()

这2个方法通过 .next\_siblings 属性对当 tag 的所有后面解析的兄弟 tag 节点进行迭代, find\_next\_siblings() 方法返回所有符合条件的后面的兄弟节点,find\_next\_sibling() 只返回符合条件的后面的第一个tag节点

find\_previous\_siblings() find\_previous\_sibling()

这2个方法通过 .previous\_siblings 属性对当前 tag 的前面解析的兄弟 tag 节点进行迭代, find\_previous\_siblings()方法返回所有符合条件的前面的兄弟节点, find\_previous\_sibling() 方法返回第一个符合条件的前面的兄弟节点

find\_all\_next() find\_next()

这2个方法通过 .next\_elements 属性对当前 tag 的之后的 tag 和字符串进行迭代, find\_all\_next() 方法返回所有符合条件的节点, find\_next() 方法返回第一个符合条件的节点

find\_all\_previous() 和 find\_previous()

这2个方法通过 .previous\_elements 属性对当前节点前面的 tag 和字符串进行迭代, find\_all\_previous() 方法返回所有符合条件的节点, find\_previous()方法返回第一个符合条件的节点